计算机视觉实验二——车道线检测

2021113117 王宇轩

1. 实验环境

- 操作系统: Windows
- 编程语言：Python

2. 文件列表

文件名	内容
image.py	图像处理部分程序
main.py	主程序
mask.png	用于生成掩膜的图像
input.mp4	输入视频
output.mp4	输出视频
实验报告.pdf	实验报告

3. 实验过程

3.1 图像处理主流程

图像处理的调用如下：

```
def process_image(image, mask):
    # 灰度转换
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    # 高斯滤波去噪
    blurred_image = gaussian_blur(gray, 9)
    blurred_image = np.uint8(blurred_image)
    # Canny边缘检测
    edges = canny(blurred_image)
    # 提取ROI
    masked_edge = ROI(edges, mask)
    # 二值化
    _, edge = cv2.threshold(masked_edge, 128, 255, cv2.THRESH_BINARY)
    # Hough直线检测
    lines, _ = lines_detector_hough(edge)
```

```
    # 画线，得到结果
    result = drawLines(lines, image, mask)
    return result
```

3.2 高斯滤波去噪

主要流程为获取高斯核并与输入图像做卷积。相关实现如下：

```
# 卷积函数
def convolution(image, kernel, average=False):
    image_row, image_col = image.shape
    kernel_row, kernel_col = kernel.shape
    output = np.zeros(image.shape)
    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)
    padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 *
pad_width)))
    padded_image[pad_height:padded_image.shape[0] - pad_height,
pad_width:padded_image.shape[1] - pad_width] = image
    for row in range(image_row):
        for col in range(image_col):
            output[row, col] = np.sum(kernel * padded_image[row:row + kernel_row,
col:col + kernel_col])
            if average:
                output[row, col] /= kernel.shape[0] * kernel.shape[1]
    return output


def dnorm(x, mu, sd):
    return 1 / (np.sqrt(2 * np.pi) * sd) * np.e ** (-np.power((x - mu) / sd, 2) / 2)


def gaussian_blur(image, size):
    sigma = math.sqrt(size)
    # 获取高斯滤波核
    kernel_1D = np.linspace(-(size // 2), size // 2, size)
    for i in range(size):
        kernel_1D[i] = dnorm(kernel_1D[i], 0, sigma)
    kernel = np.outer(kernel_1D.T, kernel_1D.T)
    kernel *= 1.0 / kernel.max()
    return convolution(image, kernel, average=True)
```

3.3 Canny边缘检测

在image.py中的主要调用为：

```
def canny(image):
    gradient_magnitude, gradient_direction = sobel_detect(image)
    image = non_max_suppression(gradient_magnitude, gradient_direction)
```

```
    weak = 50
    image = threshold(image, 5, 20, weak=weak)
    new_image = hysteresis(image, weak)
    return new_image
```

流程为：用Sobel算子获取梯度幅值和方向、进行非极大值抑制、双阈值，分离强边缘和弱边缘、连接弱边缘。具体实现详见image.py。

3.4 ROI提取

首先，通过以下函数生成mask掩膜，主要流程为读取图片、resize到所需尺寸、之后二值化。

```
# 生成mask掩膜，输入为宽、高，读取mask.png实现
def generate_mask(w, h):
    mask_img = cv2.imread('mask.png', cv2.IMREAD_GRAYSCALE)
    mask_img = cv2.resize(mask_img, (w, h))
    _, mask = cv2.threshold(mask_img, 128, 255, cv2.THRESH_BINARY)
    return mask
```

在main.py中，通过以下方式调用：

```
# 打开输入视频文件
input_video = cv2.VideoCapture(input_file)
width = int(input_video.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(input_video.get(cv2.CAP_PROP_FRAME_HEIGHT))
mask = generate_mask(w=width, h=height)
```

mask如下：

image.py中的ROI提取实现如下。

```
def ROI(image, mask):
    masked = cv2.bitwise_and(image, image, mask=mask)
    return masked
```

3.5 Hough直线检测

```
# Hough直线检测
def lines_detector_hough(edge, ThetaDim=None, DistStep=None, threshold=None,
halfThetaWindowSize=2,
                         halfDistWindowSize=None):
    row, col = edge.shape
    if ThetaDim is None:
        ThetaDim = 90
    if DistStep is None:
        DistStep = 1
    # 计算距离分段数量
    MaxDist = np.sqrt(row ** 2 + col ** 2)
    DistDim = int(np.ceil(MaxDist / DistStep))
    if halfDistWindowSize is None:
        halfDistWindowSize = int(DistDim / 50)
    # 建立投票
    accumulator = np.zeros((ThetaDim, DistDim))  # theta的范围是[0,pi). 在这里将[0,pi)进
行了线性映射.类似的,也对Dist轴进行了线性映射
    #
    sinTheta = [np.sin(t * np.pi / ThetaDim) for t in range(ThetaDim)]
    cosTheta = [np.cos(t * np.pi / ThetaDim) for t in range(ThetaDim)]
    # 计算距离（rho）
    for i in range(row):
        for j in range(col):
            if not edge[i, j] == 0:
                for k in range(ThetaDim):
                    accumulator[k][int(round((i * cosTheta[k] + j * sinTheta[k]) *
DistDim / MaxDist))] += 1
    M = accumulator.max()
    # ---------------------------------------
    # 非极大抑制
    if threshold is None:
        threshold = int(M * 1.369 / 10)
    result = np.array(np.where(accumulator > threshold))  # 阈值化
    # 获得对应的索引值
    temp = [[], []]
    for i in range(result.shape[1]):
        eight_neiborhood = accumulator[
                           max(0, result[0, i] - halfThetaWindowSize + 1):min(result[0,
i] + halfThetaWindowSize,
                                                                              
accumulator.shape[0]),
```

```
                    max(0, result[1, i] - halfDistWindowSize + 1):min(result[1,
i] + halfDistWindowSize,
                                                                      
accumulator.shape[1])]
        if (accumulator[result[0, i], result[1, i]] >= eight_neiborhood).all():
            temp[0].append(result[0, i])
            temp[1].append(result[1, i])
    # 记录原图所检测的坐标点（x,y）
    result_temp = np.array(temp)
    # -----------------------------------------------------------
    result = result_temp.astype(np.float64)
    result[0] = result[0] * np.pi / ThetaDim
    result[1] = result[1] * MaxDist / DistDim
    return result, result_temp

# 画线
def drawLines(lines, image, mask, color=(0, 255, 255), err=3):
    result = image
    Cos = np.cos(lines[0])
    Sin = np.sin(lines[0])
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            e = np.abs(lines[1] - i * Cos - j * Sin)
            if (e < err).any() and mask[i, j]:
                result[i, j] = color
    return result
```

3.6 视频处理

见main.py。将已实现的针对图像的当前车道线检测封装成一个函数`process()`，调库处理提供的视频，分解为一帧帧图像，调用函数处理，最终合成回原分辨率原帧率视频。

4. 实验结果

见output.mp4。目前可以看到视频中大部分识别正常，但也存在一些多余的误判线，调整高斯模糊卷积核大小及边缘检测的阈值能够解决这个问题，但视频处理时间不算短，繁杂的调参工作也不是实验的主要内容，我就将目前的这个结果提交了。